BRAUDE
College of Engineering, Karmiel

Software Engineering Department

Braude College

Capstone Project Phase B – 61999

Detection of Rare Coins

25-1-R-14

By Hassan Said Ahmad Hassan.Said.Ahmad@e.braude.ac.il

Lev Leyfer Lev.Leyfer@e.braude.ac.il

Supervisor: Zeev Frenkel

Github Link : www.github.com/whisper120/CoinDetector

1. Project description
    1.0 General Description
    1.1 Solution Description
    1.2 Description of the Challenges Faced and the Solutions Found
    1.3 Results and Conclusions
    1.4 Lessons Learned
    1.5 Did We Meet the Project’s Goals?
    1.6 Findings
2. User guide
    1. Launching the Application
    2. Configuring Detection Settings (Optional)
    3. Acquiring an Image
    4. Detecting Special Coins
    5. Viewing Detection Results
3. Maintenance guide
    3.1 Expanding the Dataset
    3.2. Preparing the Dataset for Training
    3.3. Retraining the Models
4. References

User documentation

1. Project description

1.0 General Description

The purpose of our project was to develop an automated system for detecting the mint year on Israeli 10 Agorot coins. The mint year is engraved on the coin in stylized Hebrew letters, which poses a significant challenge for recognition systems due to its unique texture and appearance. The system is intended for use by numismatists, collectors, and institutions looking to streamline the process of coin classification. The project includes two main phases: detection of the region containing the year and recognition of the Hebrew characters. The nature of the system is modular, and it uses AI-based image processing and recognition technologies to achieve its goals.

1.1 Solution Description

In our effort to detect mint years on Israeli coins, specifically the 10 Agorot coin we adopted a two-phase AI-driven approach. In the first phase, we trained a YOLO (You Only Look Once) model to identify the region of interest (ROI) containing the year on the coin. This part of the system was successful; the model consistently detected and cropped the section of the coin containing the year with high accuracy.

The second phase focused on recognizing the Hebrew letters printed within the cropped region. Initially, we experimented with Tesseract OCR, applying binarization techniques to reduce background noise. However, due to the metallic texture and noise inherent in the coin images, Tesseract performed poorly frequently failing to identify the majority of letters, and at best, recognizing only one letter per image.

In pursuit of better results, we trained a second YOLO model on a dataset of over 700 labeled images of Hebrew letters. This model was intended to run on the output of the first phase and detect individual letters. Unfortunately, it also produced suboptimal results. The engraved and stylized nature of the Hebrew text on the coins proved challenging for the model, which struggled to generalize from our dataset.

We also explored using a convolutional neural network (CNN) model from HuggingFace, specifically the HebrewManuscriptsMNIST [1] model, trained on a dataset of handwritten Hebrew characters. Despite the promising design of this model, it failed to deliver useful results. The network struggled to recognize the engraved letters on the coins, occasionally identifying one letter, but often incorrectly, which did not improve the overall accuracy of the system.

To improve the clarity of the cropped images, we also tried applying a super-resolution algorithm in an attempt to enhance image quality and reduce noise, hoping this would lead to better character recognition. Unfortunately, this approach yielded no noticeable improvement; the recognition results before and after applying super-resolution were essentially the same.

Initially, with only 100 images, the model's performance was poor. However, as we increased the size and diversity of the dataset, recognition accuracy improved significantly . This was a substantial improvement compared to previous attempts, and the diversity helped the model generalize across different coin types. Still, the model only sometimes identified 3–4 letters per image and frequently missed critical characters.

Because each letter is crucial for correctly identifying the mint year, partial recognition was insufficient. As a result, we decided to change our approach. Rather than continuing to expand a dataset full of highly variable conditions, we used the YOLO model to detect ROIs, then cropped these regions to create a new dataset focused solely on the engraved Hebrew letters.

We annotated these cropped images and retrained a new YOLO model on this more focused dataset. The results improved, recognition was faster and slightly more accurate, but it still did not achieve consistent, reliable full-letter recognition.

After further trial and error, we discovered **RF-DETR**, a Roboflow-developed model based on the Detection Transformer architecture. To test this model, we installed and ran it locally on our machine equipped with an NVIDIA GeForce 3060 Ti GPU. We trained RF-DETR on the same cropped and annotated Hebrew letter dataset.

The results were highly promising: RF-DETR significantly outperformed all previous models, often recognizing **every** letter in the cropped image, with only minor misses in a few cases. The combination of a YOLO-based region detector followed by RF-DETR for letter classification emerged as the most accurate and efficient solution in our experiments.

1.2 Description of the Challenges Faced and the Solutions Found

Throughout the project, we faced several significant challenges. The most critical difficulty was the accurate recognition of engraved Hebrew letters on coins. Unlike printed or handwritten text, these characters exhibit unique distortions due to their engraving style, metallic texture, background noise, and light reflections. These factors made it particularly difficult for standard OCR engines such as Tesseract and others to perform effectively, even after applying preprocessing techniques like binarization and super-resolution. While these methods provided marginal improvements, they were not sufficient on their own.

In addition, general-purpose deep learning models, including various versions of YOLO and even domain-specific datasets like HebrewManuscriptsMNIST, struggled to generalize to the distinct features of coin engravings. To address this, we built a custom dataset composed of real coin images we captured ourselves. We manually annotated these images using Roboflow and incrementally expanded the dataset through an iterative training process. We experimented with several YOLO models, gradually improving the model's ability to detect and classify the engraved Hebrew text. And finally we tried a new model of RF DETR to have better accuracy.As a result, we achieved a significant increase in recognition accuracy.

1.3 Results and Conclusions

We largely achieved the main goal of the project: building a system capable of detecting and interpreting the mint year on Israeli 10 Agorot coins. While traditional OCR methods and our early CNN-based models struggled with accuracy, especially in the presence of noise, wear, or poor lighting, our two-phase YOLO-based detection pipeline proved significantly more robust. Ultimately, the combination of **YOLO for region detection** and **RF-DETR for letter recognition** delivered the **best overall results**, offering both high accuracy and reliability.

The real breakthrough came when we shifted from relying on a small, generic dataset to building our own diverse and realistic dataset. Initially, we believed that around 100 annotated images would be sufficient, but results clearly improved as we continued to expand the dataset. Notably, we observed consistent performance gains up to approximately 300 images.

However, an unexpected issue arose: after adding another 200 images (bringing the total to 500), model performance actually **degraded**. Upon investigation, we suspect several potential causes for this decline:

- The added images may have introduced **labeling inconsistencies**, lower-quality examples, or a **distribution shift** (e.g., different lighting, camera, or coin wear levels).
- This additional data might have created a **class imbalance** or caused the model to overfit on noisy or low-relevance samples.
- It's also possible that our training pipeline did not fully adapt to the larger dataset in terms of preprocessing, learning rate, or augmentation.
- More critically, the added images lacked consistency. In an attempt to capture broader variability, we included too many heterogeneous examples, ultimately making the dataset less focused and less representative of the specific task. Instead of forming a well-defined and cohesive dataset, it became overly broad and diluted the model's

learning ability.

This outcome emphasized a valuable lesson: **more data isn't always better** unless it is **curated carefully** and consistently represents the target conditions.

We also explored different training setups. Locally, we trained the model on a desktop PC equipped with an NVIDIA RTX 3060 Ti. While this allowed us to iterate quickly, we also trained the model on Roboflow’s cloud platform, which uses industrial-grade GPUs and optimized training pipelines. Interestingly, the Roboflow-trained models consistently achieved slightly better results than our local runs.

We’re not entirely sure why that was the case, but several possible factors may explain it:

- Roboflow likely uses more **aggressive and balanced augmentation techniques**, regularization, and automatic hyperparameter tuning that improve generalization.

- Their hardware may allow for larger batch sizes and faster convergence without memory constraints.

- Their infrastructure is optimized for training vision models, possibly providing better architecture defaults, optimizer settings, and data pipeline handling than our local setup.

In summary, this project taught us that **data quality and diversity are as important as quantity**, and that **training environment and tooling** can have a notable impact on outcomes. We learned the importance of controlled experiments, consistent evaluation, and being skeptical of assumptions, even ones as intuitive as "more data = better results."

1.4 Lessons Learned

Looking back, one key lesson was the critical importance of high-quality, task-specific data. Public datasets and general-purpose models proved insufficient due to the unique visual and structural characteristics of engraved Hebrew text on coins. By shifting to a more targeted and practical data collection strategy capturing and annotating our own images we achieved significantly better results. In hindsight, we would have focused on building a custom dataset much earlier, rather than initially relying on pretrained models that weren’t well-suited to our specific use case.

Another important takeaway was the value of model and tool diversity. By experimenting with a wide range of approaches including multiple YOLO variants, custom CNNs, various OCR engines like Tesseract and the RF-DETR model, we gained a deeper understanding of the limitations and strengths of each method. This exploration eventually led us to a hybrid solution that combined the most effective components from each approach.

While the process was time consuming and involved a lot of trial and error, it was also an invaluable learning experience. We became familiar with a broad set of modern AI tools and techniques currently available in the market. The project not only improved our technical skills but also taught us how to adapt, iterate, and learn continuously along the way.

1.5 Did We Meet the Project's Goals?

Yes, we successfully met the primary goals we established at the outset of the project. Our final system achieves a very high **accuracy** level in detecting and interpreting Hebrew mint years on Israeli coins, an impressive result given the visual complexity and variability of the task. The solution demonstrates robustness across a wide range of coin conditions, including differences in lighting, wear, and image quality.

In addition to meeting the technical objectives, we also fulfilled our broader research goals: we evaluated multiple modeling strategies, iterated extensively on dataset design and quality, and explored cutting-edge AI techniques, including the RF-DETR transformer model .

By combining a YOLO-based region detector with the RF-DETR model for Hebrew letter recognition, we developed a highly effective hybrid approach. This level of exploration, combined with the accuracy and consistency of the final results, indicates that we not only met but, in many respects, exceeded the original benchmarks we set for the project.

1.6 Findings

To evaluate the performance of various models in recognizing Hebrew characters on Israeli 10 Agorot coins, we conducted an experiment using a set of 20 test images that were not included in the training data for any of the models. Each image contains a visible Hebrew year engraved on the coin.

We tested six different approaches, each employing a distinct recognition method trained on different datasets:

- **Model 1:** A YOLO model trained on an external dataset consisting of Hebrew letters written or printed on paper. This dataset is generic and does not include coin images

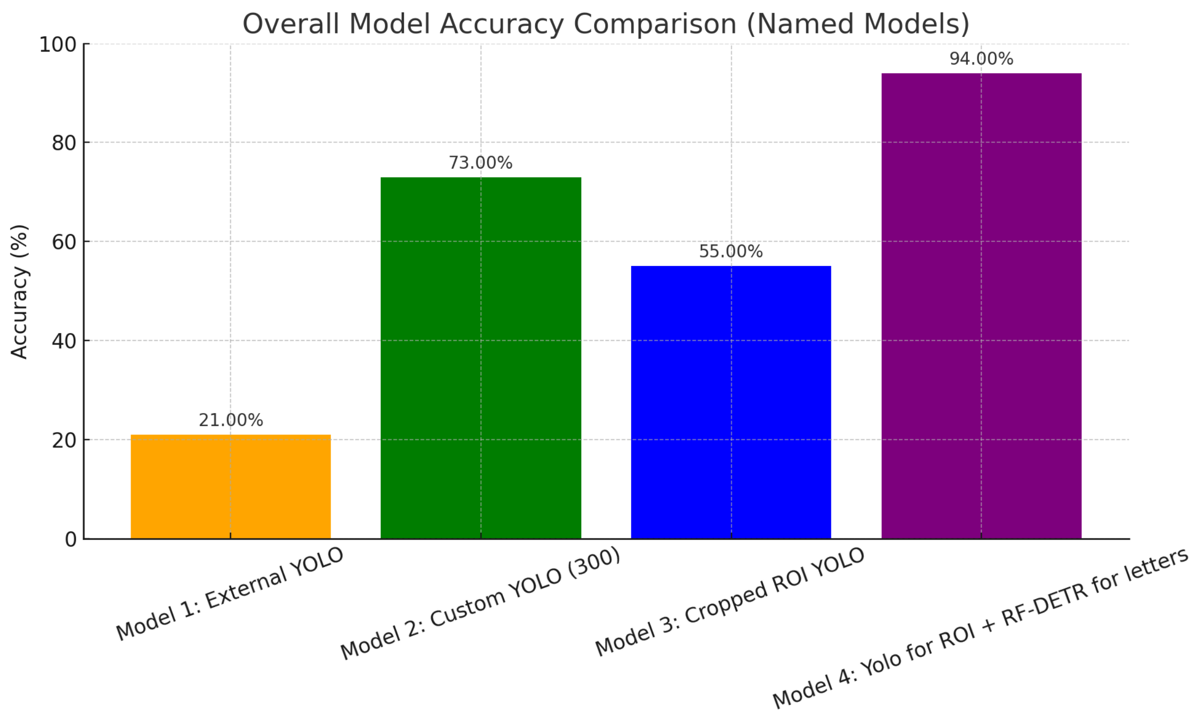
- **Model 2:** A YOLO model trained on an expanded custom dataset of 300 labeled coin images, capturing greater variability in coin conditions and lighting.

- **Model 3:** A YOLO model trained specifically on cropped images of the regions of interest (ROIs) containing Hebrew letters. These images were extracted from the coin images using the initial YOLO ROI detector, reducing background noise and focusing training on the letters themselves.

- **Model 4:** A hybrid approach combining a YOLO model trained to detect the ROI containing the year on the coin, followed by the RF-DETR model trained on the cropped letter images extracted from these ROIs. This dataset focuses specifically on the Hebrew letters engraved on coins, minimizing background noise.
- **Model 5:** The Tesseract OCR engine applied directly to the full coin images without any specialized training or adaptation..

The table shows the number of letters each model correctly identified. The ground truth was 100 letters (20 coins , each coin had 5 letters on them) , in doing so each sum of correct predictions is also a percentage

Model	Sum of Correct Predictions
Model 1 YOLO : external dataset	21
Model 2 YOLO : custom 300 images	73
Model 3: custom 500 cropped roi images	55
Model 4: Yolo for roi and RF-DETR for letters	94



Among the five tested recognition approaches, Model 4: Hybrid YOLO + RF-DETR emerged as the most accurate, achieving an 94% recognition accuracy. This method combines a YOLO detector to locate the region of interest (ROI) on the coin with a refined RF-DETR model trained specifically on cropped Hebrew letter images. Its two-step focus, first isolating relevant areas, then classifying the content, proved more effective than single-stage recognition.

Model 2: Custom YOLO trained on 300 coin images also showed strong performance, achieving 73% accuracy. This demonstrates the benefit of training on a task-specific dataset, particularly one with diverse examples of actual coins.

All models were evaluated using images captured under normal lighting conditions, with standard smartphone photography, including minor variations in angle, distance, and ambient light. These conditions reflect practical usage scenarios rather than idealized lab setups.

Key Takeaways:

- Hybrid approach (Model 4) outperforms all others due to precise ROI detection and focused classification.

- Custom-trained models (Model 2 and Model 3) significantly outperform generic solutions like Tesseract OCR.

- The YOLO model trained on external (non-coin) data performed poorly, highlighting the importance of domain-specific training.

As our project's goal is to provide the exact year of each coin, accurately recognizing all Hebrew letters in the date is essential. Among the models evaluated, only two demonstrated strong performance in this task: Model 4, a hybrid of RF-DETR and YOLO, and Model 2, a YOLO model trained on 300 custom images.

All models were tested on the same set of 20 photos that were not included in any training data. The results showed that:

- Model 3 correctly identified only 5 out of 20 complete dates.

- In contrast, Model 4 (Hybrid RF-DETR) successfully identified 14 out of 20 complete dates, outperforming all other models and demonstrating a clear advantage over YOLO-only approaches.

- For the remaining 6 photos where Model 4 did not fully identify the date, it still managed to recognize 4 out of the 5 letters, showing high partial accuracy.

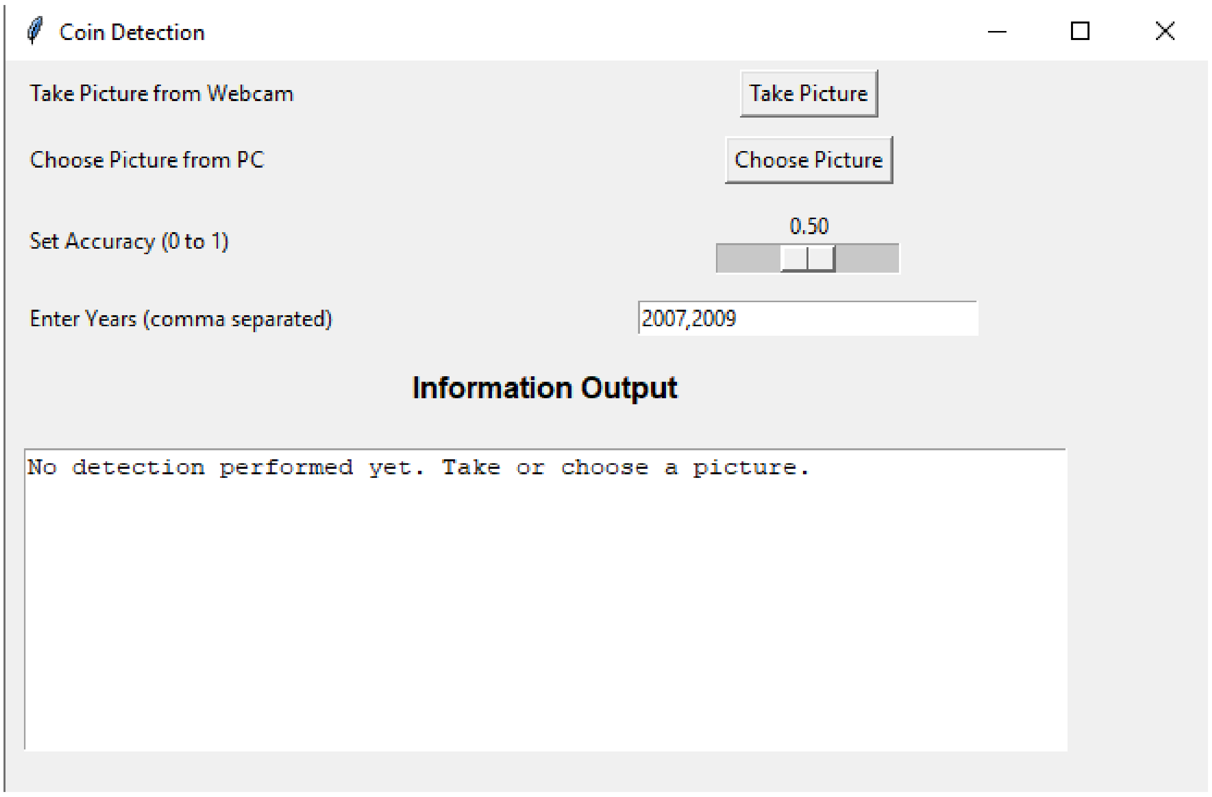
These results highlight the superiority of the hybrid approach in recognizing complete Hebrew dates on coins, which is crucial for the accuracy of our system.

2. User guide

This guide provides instructions on how to operate the system for detecting specific coins.

1. Launching the Application

Upon launching the application's Graphical User Interface (GUI), you will see the main operational window.



2007,2009

```
No detection performed yet. Take or choose a picture.
```

2. Configuring Detection Settings (Optional)

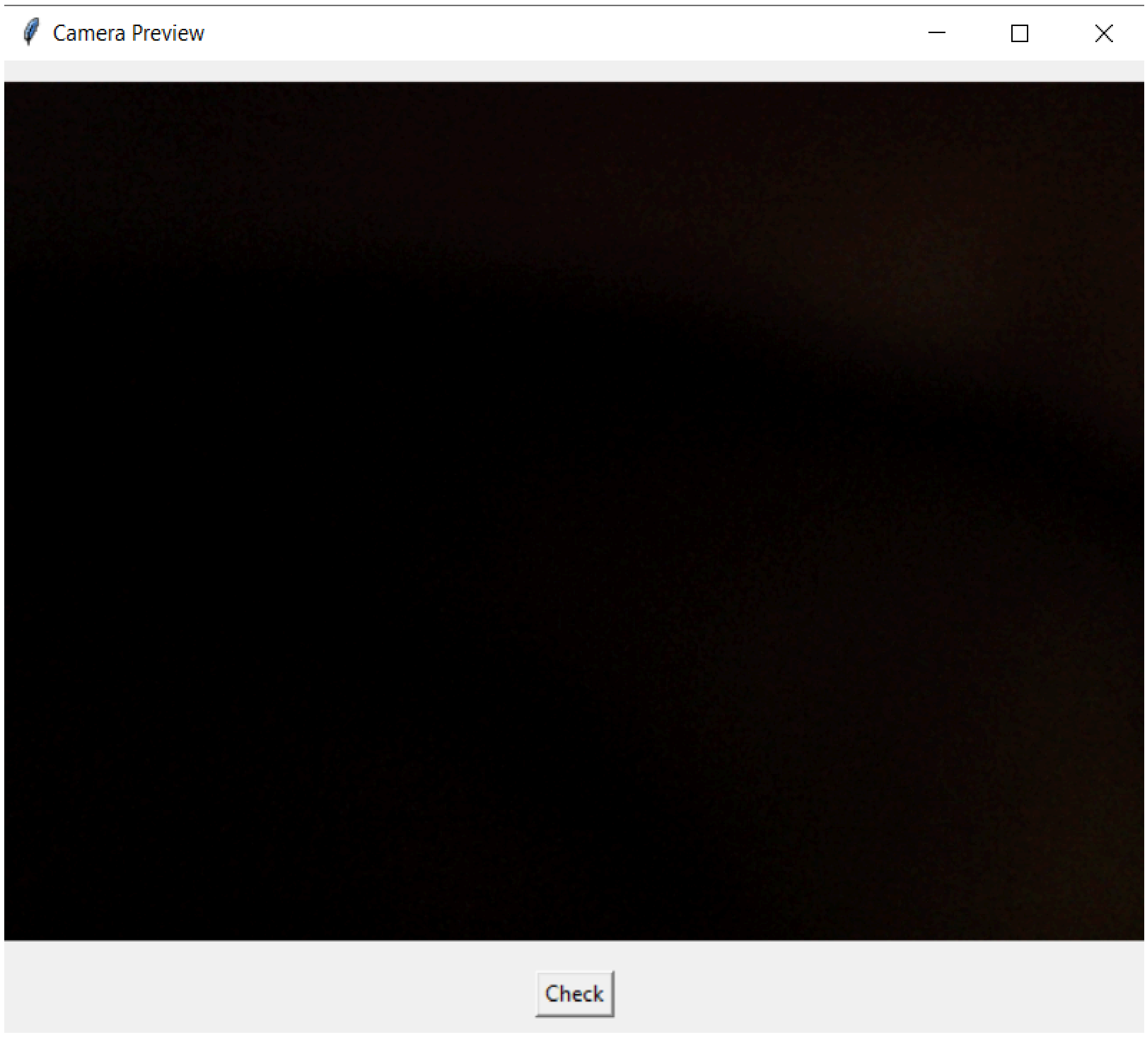
Before proceeding, you can adjust the detection settings:

- **Set Accuracy Slider:** This slider ranges from 0 to 1, with a default value of 0.5. It controls the model's confidence level for detecting letters.
  - **Recommendation:** For optimal results, it is recommended to set this slider between **0.5 and 0.8**. Setting it too high may result in missed detections, while setting it too low may produce too many irrelevant results.
- **Enter Years (separated with comma):** Below the accuracy slider, you will find an input field. Here, you can enter specific years that you want the system to notify you about if detected. Enter multiple years separated by commas (e.g., `2007,2008,2010`).

3. Acquiring an Image

You have two options to provide an image for analysis:

- **Take Picture:** Click the **"Take Picture"** button. A new window will pop up, displaying the live feed from your webcam.
- **Choose Picture from PC:** Click the **"Choose Picture"** button. This will open a file dialog, allowing you to select an image file from your computer.



4. Detecting Special Coins

1. Position your webcam to capture the coins you wish to analyze.
2. Once the coins are visible in the webcam feed, click the **"Check"** button located at the bottom of the webcam window.

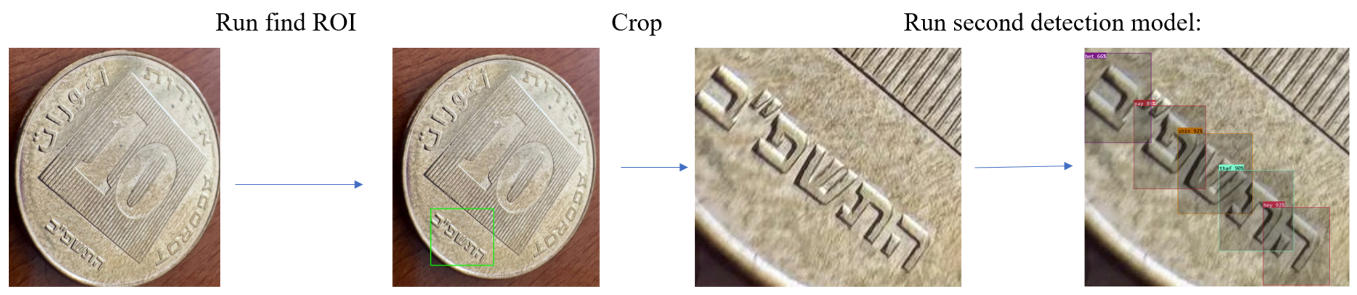
5. Viewing Detection Results

After clicking "Check," the system will process the image. Any detected coins and their relevant information will be displayed in the **"Information Output Window"** on the main GUI. This panel will provide details about the coins that have been identified, including any that match the years you specified for notification.

How does the actual system work ?

We demonstrate the process of extracting the Hebrew calendar year from coin images using a multi-stage detection pipeline. First, the system runs a Region of Interest (ROI) detection model to locate the part of the coin that contains Hebrew inscriptions (photo 1). Once the ROI is identified, it is cropped from the original image (photo 2) and passed into a secondary detection model trained specifically to recognize Hebrew characters (photo 3). The detected characters are then interpreted using a predefined algorithm that converts Hebrew letters into numerical values based on their gematria. Applying this conversion, the system successfully extracts the corresponding Gregorian year—identified in this case as 2021–2022 (photo 4).

The photos below are 1-4 from the left.



3. Maintenance guide

To enhance the system's accuracy and expand its detection capabilities (e.g., recognizing new types of coins or improving performance on existing ones), it will be necessary to update the underlying machine learning models. This involves adding more data to the dataset and retraining the models.

3.1 Expanding the Dataset

More data is essential for model improvement. Our dataset is hosted on Roboflow and also provided on GitHub.

- **Roboflow Universe:** The primary dataset is available at: https://universe.roboflow.com/levyworkspace/coin_date_database To contribute new data or modify existing annotations, simply fork this project on Roboflow, make your additions/changes, and then export the updated dataset.

- **GitHub Repository:** The dataset is also directly provided within the project's GitHub repository for convenience.

3.2. Preparing the Dataset for Training

Once new data has been added (either via Roboflow or directly to the GitHub repository), you will need to prepare it for training:

1. **Download in YOLOv11 Format:** If you updated the dataset on Roboflow, ensure you download the updated dataset in the **YOLOv11 format**. This format includes both images and their corresponding `.txt` annotation files.
2. **Organize Data:** Place the downloaded dataset into the correct folder structure as required by YOLOv11. Typically, this involves having `train`, `val`, and `test` directories, each containing `images` and `labels` subdirectories. Refer to the standard Ultralytics YOLO documentation for the precise folder structure.
3. **Update `data.yaml`:** Ensure your `datasets/data.yaml` file (e.g., `datasets/data.yaml`) is correctly configured to point to the new dataset's `train`, `val`, and `test` image and label paths, and that the class names are accurate.

   Note: for RF DETR training , you need to download from roboflow as coco format and everything else is the same.

3.3. Retraining the Models

The retraining process leverages our existing `TrainYOLO11.py` script.

1. **Locate `TrainYOLO11.py`:** This script is designed to handle the retraining of the YOLO models.
2. **Specify Previous Model Path:** In `TrainYOLO11.py`, you will need to specify the path to the **latest trained model** (e.g., `runs/detect/train/weights/best.pt`). This allows the new training run to continue from an already strong starting point (transfer learning).

Run the Training Script: Execute `TrainYOLO11.py` to initiate the training process.

```
python TrainYOLO11.py
```

3. **Replace Models:** Once training is complete and a new `best.pt` model is generated, replace the existing `yolo_letter_model` files in your application's `models/` directory with the newly trained models to update the system's detection capabilities.

Note: the same thing for RF DETR , trainRFDETR.py is the script that handles resuming training on RF DETR , just change where resume parameter to where the latest model is.

4. References

[1] https://huggingface.co/bsesic/HebrewManuscriptsMNIST

[2] https://github.com/roboflow/rf-detr